

# Trabajo 3

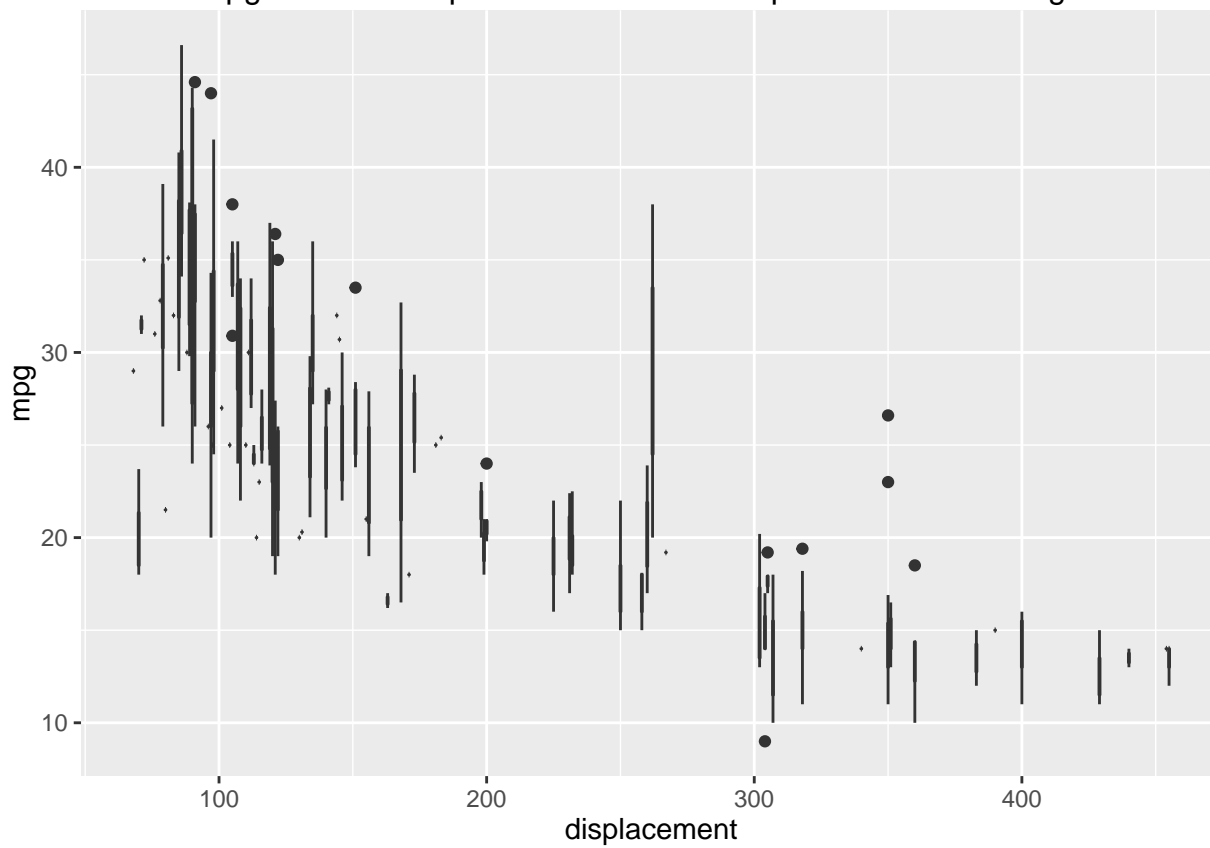
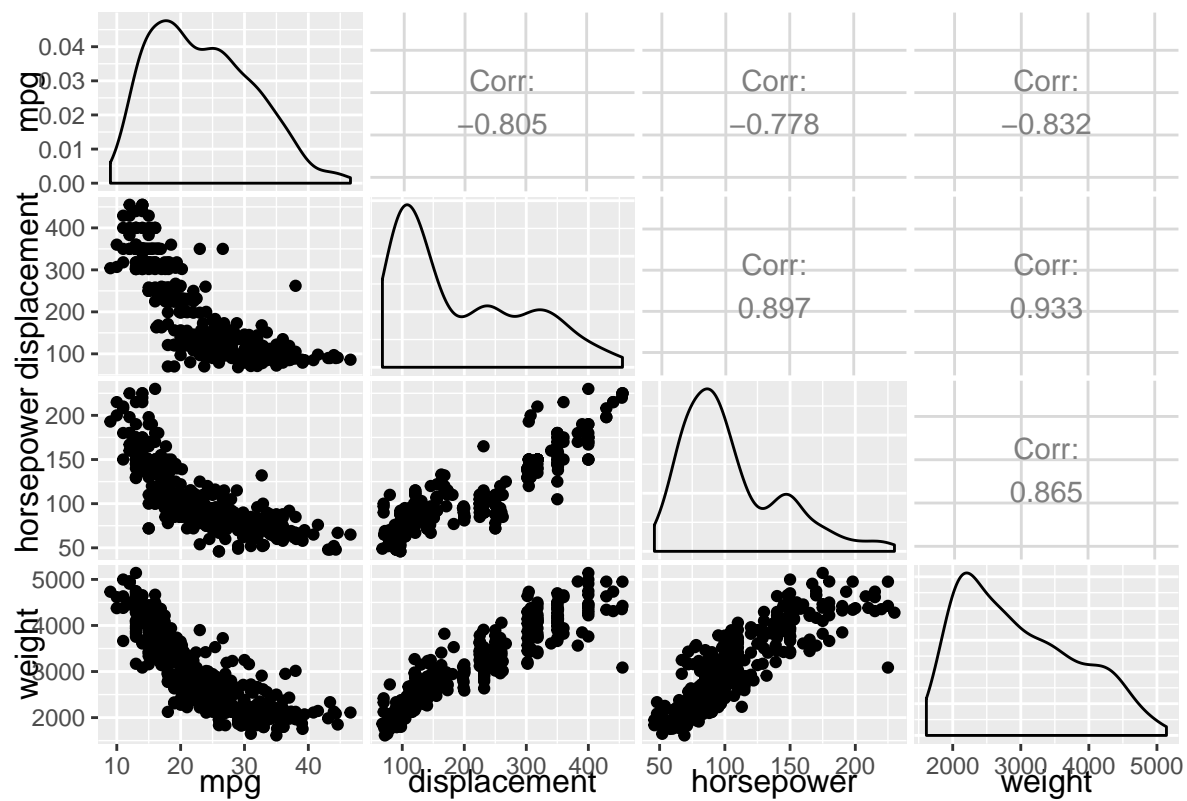
*Antonio Álvarez Caballero*

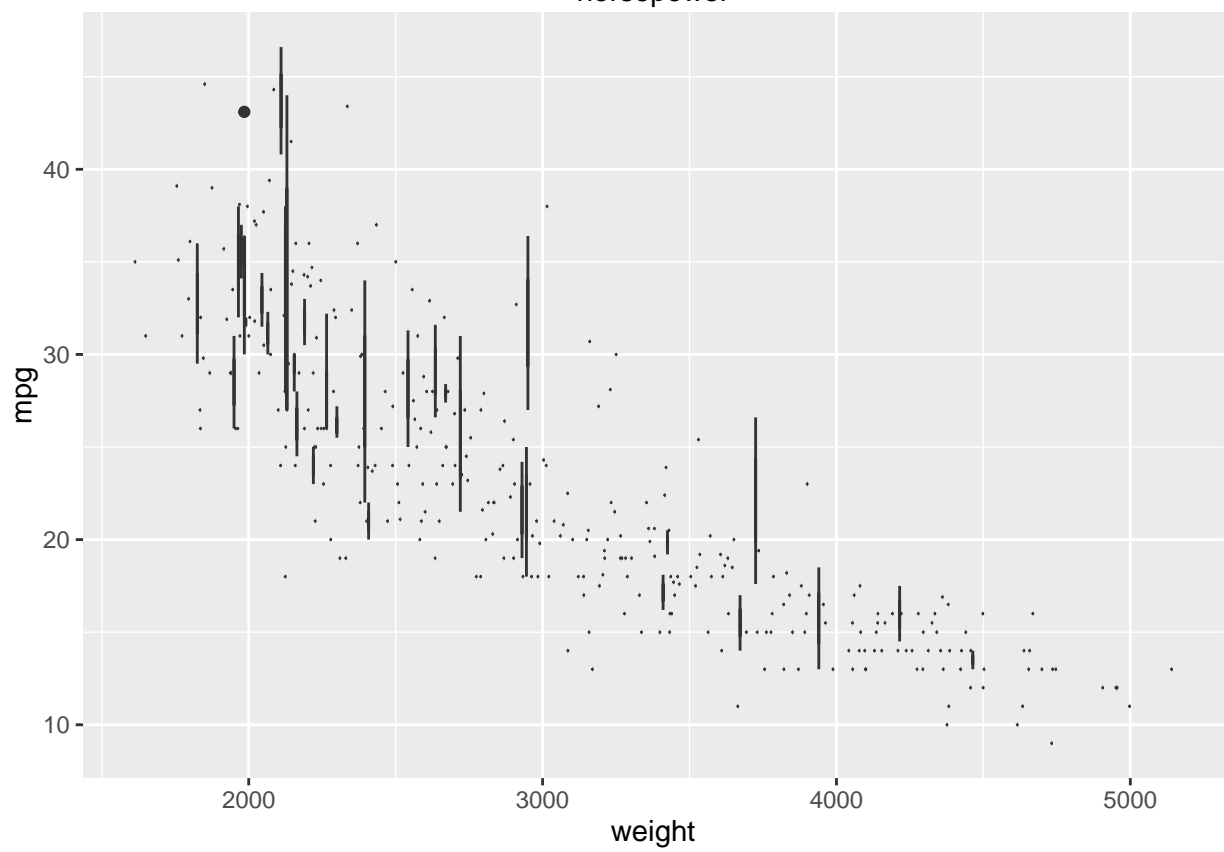
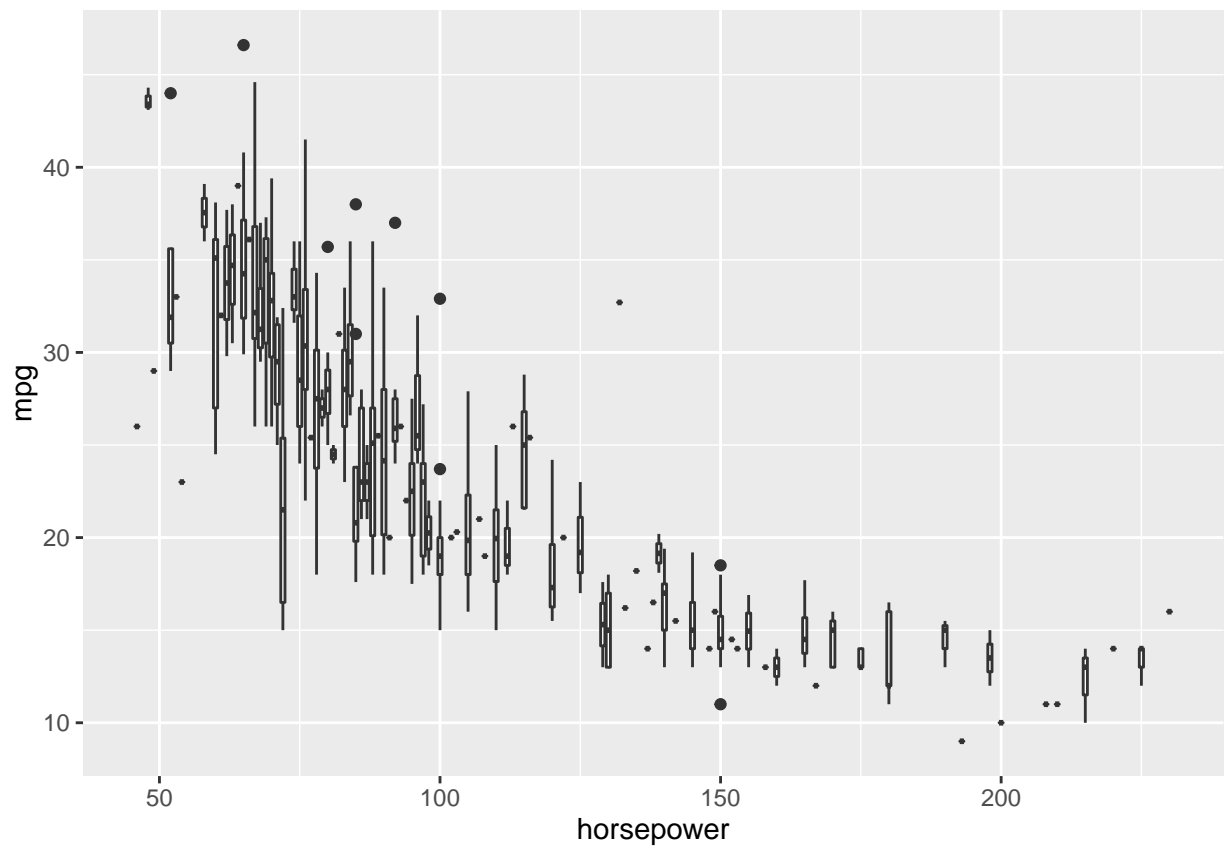
*27 de mayo de 2016*

## Ejercicio 1

### Apartado a)

Parece ser que las variables de las que más depende *mpg* son *displacement*, *horsepower* y *weight*. Veámoslas con más detalle.





## Apartado b)

Seleccionamos las variables que hemos decidido para predecir.

```
Auto.selected <- Auto[,c("displacement", "horsepower", "weight")]
```

## Apartado c)

Como nuestro conjunto de datos es grande (392 instancias), podemos realizar un muestreo aleatorio. Así tampoco falseamos las muestras, cosa que podría pasarnos si realizamos un muestreo estratificado.

```
index <- sample(nrow(Auto), size = 0.8*nrow(Auto) )
```

```
Auto.train <- Auto.selected[index,]
```

```
Auto.test <- Auto.selected[-index,]
```

## Apartado d)

Vamos a crear una nueva variable, *mpg01*, la cual tendrá 1 si el valor de *mpg* está por encima de la mediana y -1 en otro caso.

```
mpg01 <- ifelse(Auto$mpg >= median(Auto$mpg), 1, 0)
```

```
Auto.selected$mpg01 <- mpg01
```

```
Auto.train$mpg01 <- mpg01[index]
```

```
Auto.test$mpg01 <- mpg01[-index]
```

## Apartado d1)

Vamos a ajustar un modelo de regresión logística para predecir *mpg01*.

```
model.LogReg <- glm(mpg01 ~ ., data = Auto.train, family = binomial)
```

```
prediction.LogReg <- predict(model.LogReg, newdata = Auto.test)
```

```
prediction.LogReg
```

##	5	10	12	26	37
##	-4.04368525	-7.48368974	-5.44899213	-9.52300480	-1.42407833
##	38	44	45	48	49
##	-1.61152690	-8.67950592	-9.62774405	-1.79314656	-1.10221355
##	58	60	64	71	73
##	1.83211841	3.44739679	-8.13689797	-8.72066034	-5.28041034
##	79	80	87	90	92
##	0.64503893	3.07579492	-4.84599161	-5.20380250	-7.44172538
##	96	99	100	103	107
##	-11.54802619	-1.78524803	-0.93422861	4.32005837	-7.99482997
##	108	116	122	123	124
##	-0.62618624	-5.97977326	-4.45739214	0.48112296	-0.59389526
##	125	132	137	146	152
##	-6.34601345	4.17773124	-5.41012960	3.85517658	3.69981282
##	154	155	165	167	180
##	-2.31288985	-1.13603253	-1.44956306	-3.11627459	0.32691408
##	187	198	201	202	206
##	2.51972946	3.60384424	-1.62071049	-2.85040481	2.92790375
##	219	221	224	230	235

```
## 4.35179380 3.64178815 -5.75035971 -7.98131760 0.74973498
## 239 243 246 249 250
## 2.80275212 0.59960079 3.92457004 4.20408780 -2.40499017
## 258 271 277 282 287
## -1.11703835 1.13841895 0.04431415 -0.16844383 -4.21416919
## 300 318 333 341 347
## 0.54742764 2.76060074 4.06863233 0.79676274 3.37799439
## 350 353 358 359 360
## 3.56640725 2.81333105 0.93194508 1.76691942 0.16202180
## 368 372 376 378 382
## 1.43548956 1.48243855 3.48742203 3.30971798 2.80219058
## 385 388 393 397
## 3.58070768 0.86587481 0.83732702 1.33745012
```

```
error.test <- "¿Esto cómo va?"
```

El error de test de este modelo es ¿Esto cómo va?.

## Apartado d2)

Ahora vamos a ajustar un modelo k-NN.

## Apartado d3)

Veamos las curvas ROC de ambos modelos.

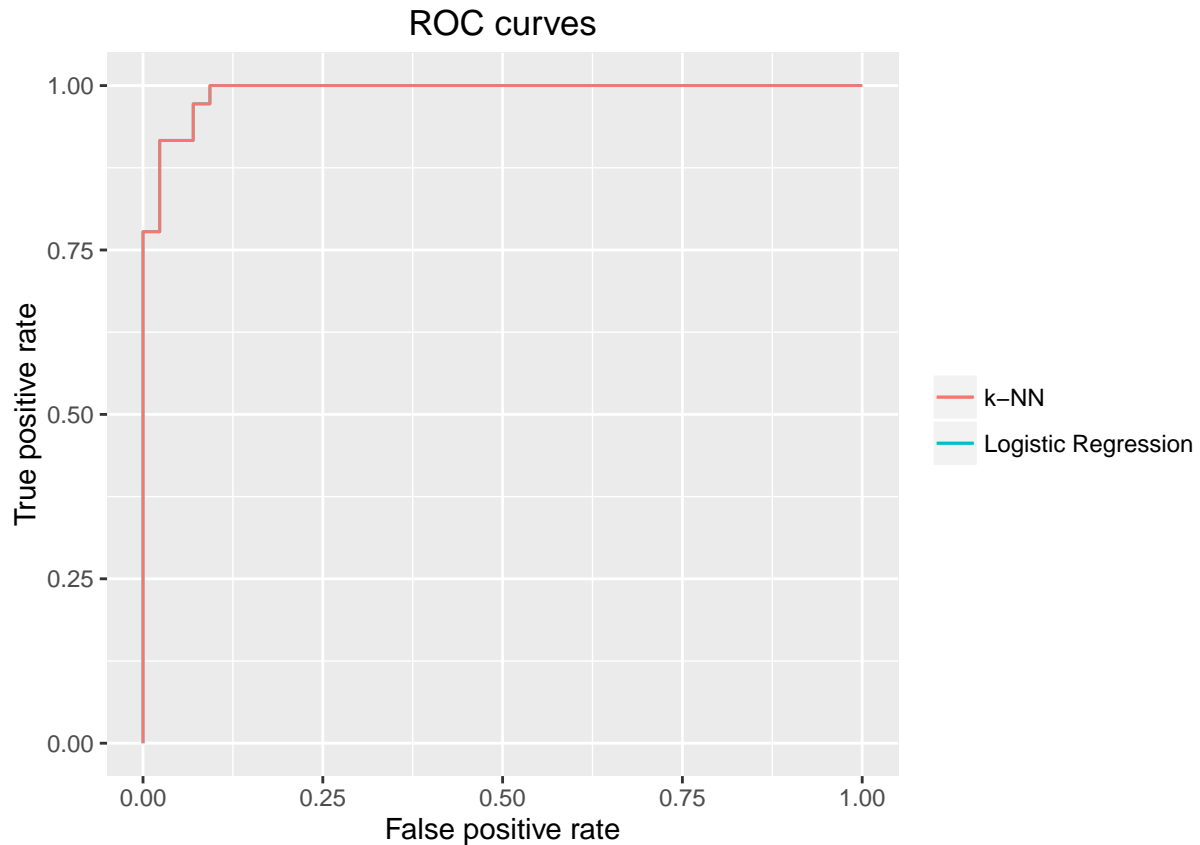
```
roc.prediction.regLog <- prediction(prediction.LogReg, Auto.test$mpg01)
roc.performance.regLog <- performance(roc.prediction.regLog, measure = "tpr", x.measure = "fpr")

### Meter del knn

####
roc.performance.knn <- roc.performance.regLog

roc.data <- data.frame(x = roc.performance.regLog@x.values[[1]],
                      y1 = roc.performance.regLog@y.values[[1]],
                      y2 = roc.performance.knn@y.values[[1]])

ggplot(roc.data, aes(x)) +
  geom_line(aes(y = y1, colour = "Logistic Regression")) +
  geom_line(aes(y = y2, colour = "k-NN")) +
  theme(legend.title = element_blank()) +
  labs(title = "ROC curves", x = "False positive rate", y = "True positive rate")
```



```
auc.regLog <- auc(roc.data$x,roc.data$y1, type = 'spline')
auc.knn    <- auc(roc.data$x,roc.data$y2, type = 'spline')
```

El área bajo la curva de la ROC de regresión logística es 0.9868717 y la del k-NN es 0.9868717. Luego k-NN es el modelo que mejor *performance* tiene.

### Apartado e) (Bonus-1)

Para estudiar el error con validación cruzada hacemos uso de `cv.glm`

```
model.full.LogReg <- glm(mpg01 ~ ., data = Auto.selected)
cv.LogReg <- cv.glm(data = Auto.selected, glmfit = model.full.LogReg, K = 5)
cv.LogReg$delta
```

```
## [1] 0.1035423 0.1033395
```

El error estimado es el primero de este vector. El segundo es un ajuste para compenar el sesgo introducido al no usar *Leave-One-Out*.

Para el caso del k-NN

Por tanto, vemos que es mejor *uno*.

### Apartado f) (Bonus-2)

Por hacer

## Ejercicio 2

### Apartado a)

Ajustamos con validación cruzada sobre la variable *crim*, que es la que está en la posición 1.

```
attach(Boston)

index <- sample(nrow(Boston), 0.8*nrow(Boston))
Boston.full <- Boston
Boston.train <- Boston[index,]
Boston.test <- Boston[-index,]

model.Boston <- glmnet(as.matrix(Boston.train[,-1]), Boston.train[,1], alpha = 1)
```

### Apartado b)

Ahora utilizamos un método LASSO y seleccionamos las variables que están por encima de un umbral.

```
cv.Boston <- cv.glmnet(as.matrix(Boston[,-1]), Boston[,1], nfolds = 5, alpha = 1)
lasso.coef <- predict(cv.Boston, type="coefficients", s = cv.Boston$lambda.min)
threshold <- 0.1
selected <- which(abs(lasso.coef) > threshold)[-1]
```

Con esto afirmamos que las características que superan nuestro umbral 0.1 son 4, 5, 6, 8, 9, 11, 13, 14.

Seguir con regularización.

### Apartado c)

Al igual que en el anterior apartado, definimos una nueva variable usando la mediana como umbral.

```
crim1 <- ifelse(Boston$crim > median(Boston$crim), 1, -1)
Boston.full$crim1 <- crim1
Boston.train <- Boston.full[index,]
Boston.test <- Boston.full[-index,]
```

Ahora ajustamos varias *SVM*, probaremos la lineal y con los núcleos disponibles, y veremos cómo se comporta cada uno. ‘Ver cuál sería mejor a priori con pairs o similar’.

```
svm.linear <- svm(crim1 ~ ., data = Boston.train[,-1], kernel = "linear")
svm.linear.prediction <- predict(svm.linear, newdata = Boston.test[,-1])
# confusionMatrix(svm.linear.prediction, Boston.test$crim1)
svm.linear.prediction
```

##	2	3	5	9	15	31
##	-0.82252184	-0.72809080	-1.00711060	-0.31060336	-0.65010314	-0.63253707
##	47	50	52	55	56	73
##	-1.02462280	-0.90718580	-0.89138476	-1.12111243	-1.06478406	-0.97076519
##	80	88	90	94	95	97
##	-0.65501325	-1.10169183	-0.95948355	-0.48362314	-0.34723009	-1.15557372
##	110	115	117	120	122	125
##	-0.57850164	-0.34903447	-0.34292145	-0.39308417	0.36498117	0.39714707
##	132	133	141	142	146	152

```
## 0.37566582 0.41862995 0.33566793 0.38481647 1.12646434 1.07520224
##      153      157      162      183      187      191
## 0.81077434 1.04963182 0.77379481 -0.80470863 -0.76717346 -0.79011222
##      199      201      204      206      211      212
## -0.90513924 -1.09438929 -0.72165449 -0.74404814 -0.63046993 -0.65598160
##      219      220      221      223      232      238
## -0.21337449 -0.21782429 -0.63465795 -0.65556938 -0.45988401 -0.47276188
##      240      251      253      258      268      270
## -0.90294523 -0.97838389 -0.90757548 0.08261912 -0.14504616 -1.04933618
##      279      284      286      290      302      305
## -0.88052403 -1.06274533 -1.25591606 -0.92761107 -0.79337605 -0.87075103
##      309      318      325      330      341      342
## -0.44492390 -0.47029723 -0.81292835 -1.10010344 -0.92749404 -1.01143364
##      345      354      356      359      360      370
## -0.79294331 -0.95419970 -1.32627595 1.08728465 1.23551558 1.10184136
##      376      377      382      386      390      391
## 0.96587257 0.97908812 0.94350282 0.98293677 0.99942824 1.03069118
##      393      394      395      397      404      411
## 0.98702676 0.97905053 0.97201087 0.99721490 0.92536606 0.85044175
##      415      416      421      423      429      431
## 1.06753281 1.05477456 1.12650835 0.87409773 0.96825447 0.80279720
##      432      440      443      445      446      455
## 0.84750299 1.11024204 1.18760188 1.14445048 1.21544873 1.18541683
##      461      467      470      471      474      494
## 1.11697821 1.04469147 0.66749769 0.78164150 0.92116178 -0.40447905
```

```
svm.polynomial <- svm(crim1 ~ ., data = Boston.train[,-1], kernel = "polynomial")
svm.polynomial.prediction <- predict(svm.polynomial, newdata = Boston.test[,-1])
# confusionMatrix(svm.polynomial.prediction, Boston.test$crim1)
svm.polynomial.prediction
```

```
##      2      3      5      9      15      31
## -0.94192116 -0.78806401 -0.84563645 -1.29723268 -0.60896262 -0.17472712
##      47      50      52      55      56      73
## -0.94238126 -0.97498086 -1.10735982 -1.13042213 -0.88473260 -0.97927064
##      80      88      90      94      95      97
## -0.77262743 -0.95726196 -0.99724612 -0.76471872 -0.76020115 -0.98326960
##      110      115      117      120      122      125
## -0.62956552 -0.70863331 -0.69608959 -0.69372255 -0.75778015 -0.58178394
##      132      133      141      142      146      152
## -0.44166134 -0.45229355 -0.17726954 0.06165511 0.92086864 1.13110871
##      153      157      162      183      187      191
## 0.16622200 0.90093560 1.05959378 -1.02002098 -0.01299808 -0.98606316
##      199      201      204      206      211      212
## -0.97764140 -0.94044625 -1.06725491 -0.77577936 0.41204440 0.23930236
##      219      220      221      223      232      238
## 0.33188924 0.40249161 0.85036421 0.55033193 -0.51829848 -0.51542943
##      240      251      253      258      268      270
## -0.91667742 -0.95912467 -0.76785795 1.88475078 0.99643027 -0.62262924
##      279      284      286      290      302      305
## -0.83563045 -3.97086788 -1.31785433 -0.80444604 -0.83195348 -0.79561371
##      309      318      325      330      341      342
## -0.75491526 -0.69949395 -0.82432723 -1.06434090 -0.79375096 -1.24754722
##      345      354      356      359      360      370
## -0.78566338 -0.81240139 -1.02763557 0.75762202 1.01724944 1.01908132
```



```
##          376          377          382          386          390          391
## 0.94906323 0.95879864 0.93649533 0.98728092 1.06059927 1.04613189
##          393          394          395          397          404          411
## 1.05926079 1.01227700 1.03436225 0.98561183 1.03473008 1.05672427
##          415          416          421          423          429          431
## 0.83674976 1.02781360 0.99803109 0.94879954 0.98977352 0.90342485
##          432          440          443          445          446          455
## 0.90568332 1.15399655 1.01293123 1.09430075 1.09120657 1.01024031
##          461          467          470          471          474          494
## 0.97163969 0.90375439 1.01096260 0.89050279 0.93608024 -0.68306846
```

```
svm.radial <- svm(crim1 ~ ., data = Boston.train[,-1], kernel = "radial")
svm.radial.prediction <- predict(svm.radial, newdata = Boston.test[,-1])
# confusionMatrix(svm.radial.prediction, Boston.test$crim1)
svm.radial.prediction
```

```
##          2          3          5          9          15          31
## -0.96593935 -0.74014673 -0.83585347 -0.86079950 0.10763094 0.97879142
##          47          50          52          55          56          73
## -0.97735549 -0.96654562 -1.18240251 -0.75787351 -0.84207195 -0.93515272
##          80          88          90          94          95          97
## -0.89763344 -1.16366669 -1.03370327 -0.88161249 -0.72991190 -1.25316314
##          110          115          117          120          122          125
## -0.29044028 -0.64300194 -0.71923508 -0.62300775 0.08470164 0.28746113
##          132          133          141          142          146          152
## 0.73279844 0.64937482 0.83744311 0.66915044 0.79055297 1.11414967
##          153          157          162          183          187          191
## 0.81872438 0.76610374 1.02837168 -0.69519688 -0.02803858 -0.94125079
##          199          201          204          206          211          212
## -0.97192757 -0.88391907 -0.96306469 -0.85759326 0.48948645 0.61291554
##          219          220          221          223          232          238
## 0.66610401 0.55883880 0.47498146 0.27616039 -0.18195290 -0.24577100
##          240          251          253          258          268          270
## -1.09648990 -0.97262598 -0.66911847 1.08410339 0.81708152 -0.25173409
##          279          284          286          290          302          305
## -1.08069222 -0.55967491 -1.04215020 -1.03245544 -1.06625820 -0.91325980
##          309          318          325          330          341          342
## -0.57701814 -0.40744934 -0.96842649 -0.96470276 -0.57161193 -1.05888848
##          345          354          356          359          360          370
## -0.96346872 -0.62397108 -0.87881765 0.93701086 1.05080023 0.93173460
##          376          377          382          386          390          391
## 0.90949453 0.94308231 0.94841143 0.93217978 0.99286737 1.03419667
##          393          394          395          397          404          411
## 0.98087457 0.98618162 0.98678216 0.98802273 0.93964003 0.80051696
##          415          416          421          423          429          431
## 0.83972788 0.90785798 1.06413284 1.04557818 1.01216337 0.95396023
##          432          440          443          445          446          455
## 0.91142396 1.03134551 1.06703788 1.03962790 0.96338609 0.92000864
##          461          467          470          471          474          494
## 1.08293144 0.99575619 0.89954051 0.97275270 1.01153053 -0.41844807
```

```
svm.sigmoid <- svm(crim1 ~ ., data = Boston.train[,-1], kernel = "sigmoid")
svm.sigmoid.prediction <- predict(svm.sigmoid, newdata = Boston.test[,-1])
# confusionMatrix(svm.sigmoid.prediction, Boston.test$crim1)
svm.sigmoid.prediction
```

```
##          2          3          5          9         15
## -0.05945781 -0.74035688 -3.29148884 -3.95870979 -2.21952959
##          31         47         50         52         55
## -7.00574408 -0.60282240 -1.16406446  1.14832797  5.98342552
##          56         73         80         88         90
## -15.96743858  0.72668613  0.04106608 -0.52991545 -0.02886361
##          94         95         97        110        115
##  4.59546721  3.83874296 -1.14818566 -3.73697867 -2.04687325
##         117        120        122        125        132
## -1.45442104 -2.26368625 -0.42289140 -1.12929415  0.57145123
##         133        141        142        146        152
##  0.62016954  0.81601218  6.00879382 -5.32884132 -3.21365465
##         153        157        162        183        187
## -6.00512167 -5.17549212 10.94711347  1.79531114 -2.05154228
##         191        199        201        204        206
## -7.49981630 -16.03488215 -13.16570482 -18.13622075  0.40955487
##         211        212        219        220        221
## -1.01986854 -3.94679757 -1.52904791  1.75056916  1.66247828
##         223        232        238        240        251
##  1.99518343  0.65047796  0.61356533 -0.90782245  0.07637239
##         253        258        268        270        279
## -4.05286330  9.42135654  4.42726858  4.39898716  1.18377126
##         284        286        290        302        305
## -19.11848748 -5.05063078 -3.30160561  0.38396171 -1.29092787
##         309        318        325        330        341
##  1.22109437 -1.88759862  0.56171033 -1.16946099 -1.02185250
##         342        345        354        356        359
## -7.82995406 -2.29635587 -15.62018211  2.80066557 -5.35268726
##         360        370        376        377        382
## -0.30298211 -2.76560278 -3.29679334  1.46364829  2.14391814
##         386        390        391        393        394
## 11.91427764  7.00035325  3.50132839  9.88493413  1.01838333
##         395        397        404        411        415
##  2.85696579  1.97857328  6.90633420  1.48625580 18.59697191
##         416        421        423        429        431
##  9.32104354  0.49278115 -1.19661726  3.35020638 -1.16459688
##         432        440        443        445        446
## -1.44300697  6.91653071  1.77839795  7.87140145  6.92099424
##         455        461        467        470        471
##  2.26214613 -1.30574793  0.81052246 -3.73009159 -3.31998240
##         474        494
## -4.99372860 -1.26422621
```

Analizar el error.

## Apartado d) (Bonus-3)

Ajustamos con validación cruzada sobre la variable *crim*.

```
cv.Boston <- cv.glmnet(as.matrix(Boston[,-1]), Boston[,1], nfolds = 5, alpha = 1)
```

El error de validación cruzada (respondiendo al Bonus-3) es

```
cv.Boston$cvm
```

```
## [1] 73.18910 70.02364 66.63692 63.22860 60.39827 58.04786 56.09593
## [8] 54.47489 53.01730 51.70605 50.56053 49.54884 48.68732 47.98180
## [15] 47.40236 46.92175 46.52106 46.18694 45.90859 45.67129 45.45625
## [22] 45.24990 45.05418 44.88901 44.74981 44.63289 44.53279 44.44685
## [29] 44.37432 44.30845 44.24058 44.14885 44.04908 43.94954 43.86372
## [36] 43.80016 43.75173 43.69488 43.62922 43.55415 43.47634 43.41514
## [43] 43.36609 43.32745 43.28323 43.23711 43.20371 43.17616 43.15469
## [50] 43.13743 43.12356 43.11371 43.10480 43.09708 43.09161 43.08766
## [57] 43.07883 43.07458 43.06981 43.06786 43.06547 43.06394 43.06274
## [64] 43.06208 43.06249 43.06266 43.06300 43.06391 43.06457 43.06525
## [71] 43.06684 43.06781 43.06850 43.06953 43.07003 43.07066
```